Parsing INI Files in Lisp

Gene Michael Stover

created Sunday, 2005 February 20
updated Sunday, 2005 February 20

Contents

1 What is this?

2 INI syntax

An INI file is a text file containing a sequence of *sections*. Each section (with the possible exception of the first section) begins with a header, which names the section. The label is a string of characters delimited between [brackets]. The first section need not have a label; in this case, the parser will assume that section's label is the empty string. All sections after the first *must* have a label.

After the label, a section contains attribute-value pairs. An attribute is a string, a value is a string, & they are separated (or joined) by an equal sign (=).

Labels, attributes, & values can have almost any length except zero. They may contain any text character except those that form end-of-line sequence in the local computer's convention. In other words, on, say, Windows, labels may

```
attr0=val0
attr1=val1
attr2=val2 containing multiple words
```

Figure 1: An example INI file containing just one section, & that sectin has no label

not contain a carriage return or a line feed. On unix[1], the end-of-line sequence is just one character, the carriage return. On Macintosh, the end-of-line sequence is just one character, the line feed. To be portable back to Windows, both unix & Macintosh should exclude both carriage return & line feed from labels, attributes, & values.

INI files may have comments. A comment begins with a semicolon & extends to the end-of-line. The end-of-line is not part of the comment. Comments may start at any column in the line. Labels, attributes, & values may not include semicolons.

Here is a formal grammar for INI files. Because I did not use a parser generator for this project, I cannot be sure that this grammar is correct.

$$S \rightarrow [body]section - list \tag{1}$$
$$section - list \rightarrow section - listsection \tag{2}$$
$$section \rightarrow labelbody \tag{3}$$
$$label \rightarrow \text{“}[''string\text{“}]''eol \tag{4}$$
$$body \rightarrow attr - value - list \tag{5}$$
$$attr - value - list \rightarrow attr - value - listattr - value \tag{6}$$
$$attr - value \rightarrow string\text{“} ='' stringeol \tag{7}$$
$$string \rightarrow (sequenceofanycharactersexcepteol) \tag{8}$$
$$eol \rightarrow (end - of - linecharacter(s)) \tag{9}$$

3 Example INI files & their translations

Let's look at some example INI files & the Lisp data I would like the parser to produce for each of them.

Figure 1 shows an example INI file. It contains just one section, & that section does not have a label.

The INI file in Figure 1 contains one section without a label. That section defines three attribute-value pairs. The first attribute is “attr0”, & its value is “val0”. The second attribute is “attr1”, & its value is “val1”. The third attribute is “attr2”, & its value is “val2 containing multiple words”.

[1]More specifically, on unix, POSIX, & other unix-like systems, including BSD, AIX, DYNIX, . . . & yes, Linux.

```
(("" ; section's label
;; Order of the attr-val pairs is unpredictable
("attr0" . "val0")
("attr1" . "val1")
("attr2" . "val2 containing multiple words")))
```

Figure 2: The Lisp expression that should result from compiling the INI file in Figure 1

```
;;; Second example
;;; Contains comments & empty lines
;;; which the parser will ignore.
[section0] attr0.0=val0.0
attr0.1=val0.1
attr0.2=val0.2 ; still in section 0
[section1] attr1.0=val1.0
attr1.1=val1.1
```

Figure 3: An example INI file containing two labeled sections, comments, & some empty lines

If I fed Figure 1, I'd like it to return a Lisp datum as I've shown in Figure 2.

Right off the bat, I notice that, while the list-of-lists structure I've specified here might be great for generic manipulation, more basic use of it, in which a program just wants to see what value is specified for a particular program parameter, might be clumsy. We could write some functions that would make that use of INI files more convenient. So I guess we'll stick with this list-of-lists format for now.

Figure 3 is a more complex example. It contains two named sections & also some comments.

In Figure 3, the first three lines are comments; they will be ignored entirely. The blank lines will be ignored, too. Notice that the blank line between "attr0.1" & "attr0.2" does not change to a new section. We don't change to a new section until we find a section label.

Figure **??** shows the Lisp expression that the INI file parser should return for Figure 3.

Notice that the Lisp expression in Figure 4 does not contain the blank lines or the comments from Figure 3.

4 Parsing Strategy

From the examples in Section 3, I'd say the basic parsing algorithm should be:

1. Create a list of sections to return.

```
(("section0" ; section's label
;; Order of the attr-val pairs is unpredictable
("attr0.0" . "val0.0")
("attr0.1" . "val0.1")
("attr0.2" . "val0.2")) ("section1"
("attr1.0" . "val1.0")
("attr1.1" . "val1.1"))
```

Figure 4: The Lisp expression that should result from compiling the INI file in Figure 3

2. Create a current section whose name is the empty string.
3. Until end-of-input
   (a) Get the next line of input.
   (b) Strip any comments from it. Strip leading & trailing spaces from it.
   (c) If the line is an attribute-value pair, extract the attribute & value. Append them to the current section.
   (d) Else if the line is a label, create a new section, use the label to name it, & push it onto the list of sections.
   (e) Else if it's an empty line (because we already stripped the comments, leading space, & trailing space), ignore it.
   (f) Else, we have a problem.
4. Return the list of sections.

You know, if the lexer recognized the kind of line we have, the algorithm would be even simpler. Maybe the lexer could get the next line. It could automagically strip comments & skip over empty lines. So it would never return an empty line; it would silently gobble them up. It could return a single string for label lines, a dotted pair for attribute-value lines, & a special value for end-of-input.[2] The lexer might need to return something else to indicate error.

With such a lexer function, the parser algorithm simplifies to this:

1. Create a list of sections to return.
2. Create a current section whose name is the empty string.
3. Until end-of-input
   (a) Get the next token of input.

[2] I find that the best value to return to indicate end-of-input on a stream is the stream itself. That's a safe value to signify end-of-input because you know a stream can't contain itself. At least it can't most of the time.

(b) If the token is a string, create a new section with the string as its name, push it onto the list of sections, & use it as the current section.

(c) If the token is a CONS, append it to the current section.

(d) Otherwise,we have an error.

4. Return the list of sections.

5 Top-Level API

The top-level function will be LOAD-INI. Given the pathname for an INI file, it returns the parsed contents of that file. Here is the Lisp source code for LOAD-INI:

```
(defun load-ini (pathname)
  (with-open-file (strm pathname)
    (read-ini strm)))
```

LOAD-INI is super-simple because it passes the buck to READ-INI.

```
(defun read-ini (strm)
  (let ((lst (("")))
    (do ((x (lex strm) (lex strm)))
        ((eq x strm))
        (typecase x
         (string
           ;; New section
           (push (list x) lst))
         (cons
           ;; Append this pair to the current
           ;; section.
           (setf (first lst)
                 (append (first lst) (list x))))
         (otherwise
           ;; Error
           (cerror "~&~A" x))))
    lst))
```

READ-INI needs one other function, LEX. LEX is the lexical analyzer. It returns a single string for labels, a dotted pair for attribute-value pairs, or the stream for end-of-input. It never returns comments or empty lines, & it has to recognize syntax errors.

```
(labels
  ((lex-end-of-input (line strm)
     (and (eq line strm) strm))
   (lex-label (line)
```

```
    (and (>= (length line) 3)
         (equal (char line 0) #\[)
         (equal (char line (1- (length line))) #\])
         (subseq line 1 (1- (length line)))))
  (lex-attribute-value (line)
    (and (find #\= line) ; contains =
         (plusp (find #\= line)) ; the = is after first column
         (cons
           (subseq line 0 (1- (find #\= line)))
           (subseq line (1+ (find #\= line)))))))
(defun lex (strm)
  ;; The NEXT-LINE function removes comments &
  ;; skips blank lines.  So it returns a line
  ;; with something in it, or STRM on end-of-input.
  (let ((x (next-line strm)))
    (or (lex-end-of-input x strm)
        (lex-label x)
        (lex-attribute-value x)
        'ini-lex-error))))
```

Function LEX uses a bunch of helpers, but I've defined most of them in a LABELS which wraps LEX. I hope they are self-explanatory. Each uses AND instead of IF to check conditions &, if they are all true, finally to return the desired value. Otherwise, they return NIL.

```
(defun next-line (strm)
  ;; Get a line from the stream...
  (do ((x (strip (read-line strm nil strm))
          (strip (read-line strm nil strm))))
      ;; ...until we find a non-empty string, or
      ;; a non-string.
      ((and (stringp x) (equal x ""))
       x)))
```

NEXT-LINE uses a function called STRIP to remove comments & excess spaces. If STRIP's argument is not a string, it doesn't puke; it just returns that argument unmodified.

```
(defun strip (x)
  (if (stringp x)
      (string-trim
        " "
        (if (find #\; x)
            ;; There is a comment character, so
            ;; remove it & everything after it.
            (subseq x 0 (1- (find #\; x)))
          ;; There's no comment, so don't
```

```
      ;; truncate the string.
      x))
x))
```

6 What about a parser generator?

I decided to hand-code the INI file parser, as you've seen, because it's a simple job, & a full-blown compiler compiler would be over-kill. Curiosity might force me to implement an INI file parser with a compiler compiler some other time.

A Other File Formats

- This document is available in multi-file HTML format at http://lisp-p.org/pil/.
- This document is available in Pointless Document Format (PDF) at http://lisp-p.org/pil/pil.pdf.

References